Remembering Without Memory: Tree Exploration by Asynchronous Oblivious Robots

Paola Flocchini, David Ilcinkas, Andrzej Pelc, Nicola Santoro

Remembering Without Memory: Tree Exploration by Asynchronous Oblivious Robots

Paola Flocchini[a,1], David Ilcinkas[*,b,2], Andrzej Pelc[c,1], Nicola Santoro[d,1]

[a] *SITE, University of Ottawa, Canada*
[b] *LaBRI, CNRS & Université de Bordeaux, France*
[c] *Département d'informatique, Université du Québec en Outaouais, Canada.*
[d] *School of Computer Science, Carleton University, Canada*

Abstract

In the effort to understand the algorithmic limitations of computing by a swarm of robots, the research has focused on the minimal capabilities that allow a problem to be solved. The weakest of the commonly used models is ASYNCH where the autonomous mobile robots, endowed with visibility sensors (but otherwise unable to communicate), operate in Look-Compute-Move cycles performed asynchronously for each robot. The robots are often assumed (or required to be) oblivious: they keep no memory of observations and computations made in previous cycles.

We consider the setting when the robots are dispersed in an anonymous and unlabeled graph, and they must perform the very basic task of *exploration*: within finite time every node must be visited by at least one robot and the robots must enter a quiescent state. The complexity measure of a solution is the number of robots used to perform the task.

We study the case when the graph is an arbitrary tree and establish some unexpected results. We first prove that, in general, exploration cannot be done efficiently. More precisely we prove that there are n-node trees where $\Omega(n)$ robots are necessary; this holds even if the maximum degree is 4. On the other hand, we show that if the maximum degree is 3, it is possible to explore with only $O(\frac{\log n}{\log \log n})$ robots. The proof of the result is constructive. We also prove that the size of the team used in our solution is asymptotically *optimal*: there are trees of degree 3, whose exploration requires $\Omega(\frac{\log n}{\log \log n})$ robots. Our final result shows that the difficulty in tree exploration comes in fact from the symmetries of the tree. Indeed, we show that, in order to explore trees that do not have any non-trivial automorphisms, 4 robots are always sufficient and often necessary.

Key words: mobile agent, robot, oblivious, asynchronous, tree, exploration

*Corresponding author
Université de Bordeaux, LaBRI UMR 5800
351 cours de la Libération
F-33405 Talence Cedex, France
Phone: (+33) 540 006 912. Fax: (+33) 540 006 669.
Email addresses: flocchin@site.uottawa.ca (Paola Flocchini), david.ilcinkas@labri.fr (David Ilcinkas), pelc@uqo.ca (Andrzej Pelc), santoro@scs.carleton.ca (Nicola Santoro)

[1] Partially supported by NSERC Discovery grant. Andrzej Pelc is also partially supported by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

[2] This work was done during the stay of David Ilcinkas at the Research Chair in Distributed Computing of the Université du Québec en Outaouais and at the University of Ottawa, as a postdoctoral fellow. Research partially supported by the ANR project "ALADDIN", and the INRIA project "CEPAGE".

1. Introduction

1.1. The Problem and the Model

An important goal of theoretical research on computing by autonomous mobile robots has been to understand the algorithmic limitations of computing in such settings. The research has thus focused on the minimal capabilities that allow a problem to be solved by a swarm of robots. In the investigations, three models are commonly used: SYNCH, SSYNCH, and ASYNCH; the (fully) synchronous model SYNCH being the strongest, the asynchronous model ASYNCH being the weakest, the semi-synchronous model SSYNCH lying in between (see, e.g., [3, 11, 12, 13, 22, 27, 29, 30, 31]).

In ASYNCH the autonomous mobile robots are endowed with visibility sensors (but otherwise unable to communicate), are anonymous, are oblivious, and operate in asynchronous Look-Compute-Move cycles. In one cycle, a robot uses its sensors to obtain a snapshot of the current configuration (Look); then, based on the perceived configuration, it computes a destination (Compute), and moves there (Move); if the destination is the current position, the robot is said to perform a null move. Cycles are performed asynchronously for each robot; this means that, even if the operations are instantaneous, the time between Look, Compute, and Move operations is finite but unbounded, and is decided by the adversary for each action of each robot. The robots are oblivious: they keep no memory of observations and computations made in previous cycles. All robots are identical and execute the same algorithm. It is usually assumed that robots are capable of multiplicity detection: during a Look operation, a robot can determine if at some location there are no robots, there is one robot, or there are more than one robots; however, in the latter case, the robot might not be capable of determining the exact number of robots.

To cooperatively perform an assigned task, the robots must thus overcome the several severe limitations imposed by the model, in particular their inability to communicate explicitly and to remember the past. The asynchrony of their behavior is another crucially hindering factor.

The asynchrony implies that a robot $\mathcal{R}$ may observe the position of the robots at some time t; based on that observation, it may compute the destination at some time $t' > t$, and Move to its destination at an even later time $t'' > t'$; thus it might be possible that at time t'' some robots are in different positions from those previously perceived by $\mathcal{R}$ at time t, because in the meantime they performed their Move operations (possibly several times). In other words, robots may move based on significantly outdated perceptions, which adds to the difficulty of exploration. Since robots are oblivious, i.e., they do not have any memory of past observations, the destination is decided by a robot during a Compute operation solely on the basis of the location of other robots perceived in the previous Look operation.

In the literature, the ASYNCH model is used by researchers in the study of the coordination and control of autonomous mobile robots in the two-dimensional plane, which we shall term the *continuous* scenario. The computational capabilities of these robots when the spacial universe is a network or a graph, a scenario that we shall term *discrete*, has been recently investigated in [20, 25], where the graph is a ring. In the discrete scenario, the computed destination in each cycle is either the node where the robot is currently situated or a node adjacent to it.

An important feature of the discrete scenario is the fact that the graph is totally anonymous: not only nodes but also edges are unlabeled, and there are no port numbers at nodes. This gives additional power to the adversary at the time when a robot must move. Indeed, it may happen that two or more edges incident to a node v currently occupied by the deciding robot look identical in the snapshot obtained during the last Look action , i.e., there is an automorphism of the tree which fixes v, carries empty nodes to empty nodes, occupied nodes to occupied nodes, and multiplicities to multiplicities, and carries one edge to the other. In this case, if the robot decides to take one of the ports corresponding to these edges, it may take any of the identically looking ports. We assume the worst-case decision in such cases, i.e., that the actual port among the identically looking ones is chosen by an adversary. This is a natural worst-case assumption and our algorithm is also resistant to such adversarial decisions. This assumption is also used in impossibility arguments: in some cases the adversary may prevent exploration by directing a robot to an already explored part of the tree, instead of the yet unexplored part.

We continue the study of computational capabilities of robots under the discrete scenario by considering the very basic task of *exploration*: within finite time every node must be visited by at least one robot and

the robots must enter a quiescent state. The complexity measure of a solution is the number of robots used to perform the task. The problem of exploring a graph has been extensively studied in the literature under a variety of assumptions (e.g. see [2, 15, 19, 24, 26]) but not in the setting considered here. The only exception is [20] where we proved that the minimum number $\rho(n)$ of robots that can explore a ring of size n is $O(\log n)$ and that $\rho(n) = \Omega(\log n)$ for arbitrarily large n.

1.2. Our results

In this paper we consider the case when the graph is an arbitrary tree and establish some unexpected results. We first prove that, in general, exploration cannot be done efficiently. More precisely we prove that there are n-node trees where $\Omega(n)$ robots are necessary; this holds even if the maximum degree is 4. We then prove the existence of a *complexity gap*. We show that if the maximum degree of the tree is 3 then it is possible to explore it with only $O(\frac{\log n}{\log \log n})$ robots. The proof of the result is constructive. We design an algorithm that allows a team of asynchronous oblivious robots of that size, arbitrarily dispersed among the nodes of an arbitrary tree of maximum degree three (no more than one robot per node), to explore the tree and terminate. In our setting, the major difficulty in the design of a terminating exploration algorithm comes from the fact that, because of obliviousness, the robots must always form different configurations until the task is completed (otherwise the adversary will force the robots in an endless cycle without completing the task). We overcome this difficulty. In fact, even without communication and without persistent memory, the robots are able, in our solution, to implement a distributed memory with several counting modules (the "brain") that enables them to perform the task and to terminate.

We also prove that the size of the team used in our solution is asymptotically *optimal*: there are trees of degree 3, whose exploration requires $\Omega(\frac{\log n}{\log \log n})$ robots.

Our final result shows that the difficulty in tree exploration comes in fact from the symmetries of the tree. Indeed, we show that, in order to explore trees that do not have any non-trivial automorphisms, 4 robots are always sufficient and often necessary.

1.3. Related Work

Algorithms for graph exploration by mobile agents (robots) have been recently studied by many authors. Most of the research is concerned with the case of a single robot exploring the graph. In [2, 7, 8, 14] the robot explores strongly connected directed graphs and it can move only in the direction from tail to head of an edge, not vice-versa. Many papers, e.g., [15, 16, 17, 26] study the scenario where the explored graph is undirected and the robot can traverse edges in both directions. In [15] the authors investigate the problem of how the availability of a map influences the efficiency of exploration. In some papers, additional restrictions on the moves of the robot are imposed. It is assumed that the robot has either a restricted tank [6, 9], forcing it to periodically return to the base for refueling, or that it is tethered, i.e., attached to the base by a rope or cable of restricted length [17].

In the case of anonymous graphs it is impossible to explore arbitrary graphs by a single robot, if no marking of nodes is allowed and the size of the graph is unknown. (In fact, even the family of all rings cannot be explored in this case.) Hence the scenario adopted in [7, 8] allows the use of *pebbles* which the robot can drop on nodes to recognize already visited ones, and then remove them and drop in other nodes. The focus is on the minimum number of pebbles allowing efficient exploration and mapping of arbitrary directed n-node graphs. (In the case of undirected graphs, one pebble suffices for efficient exploration.) In [8] the authors compare exploration power of one robot with a constant number of pebbles to that of two cooperating robots, and give an efficient exploration algorithm for the latter scenario. In [7] it is shown that one pebble is enough if the robot knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise.

In all the above papers, except [8], exploration is performed by a single robot. Exploration by many robots has been investigated mostly as a graph optimization problem, in the context when moves of the robots are centrally coordinated. In [18], approximation algorithms are given for the collective exploration problem in arbitrary graphs. In [4, 5] the authors construct approximation algorithms for the collective exploration problem in weighted trees. On the other hand, in [23] the authors study the problem of distributed collective

exploration of trees of unknown topology. However, the robots performing exploration have memory of all past actions and can directly communicate with each other.

The very weak robots considered in this paper have been first employed in the discrete scenario (i.e., graphs) to study the problem of gathering robots in one location [25]. These weak robots are commonly used by researchers in the study of the coordination and control of autonomous mobile robots in the continuous scenario (i.e., two-dimensional plane) [1, 3, 10, 11, 12, 13, 21, 22, 27, 29, 31]; for a recent survey see [28]. The continuous scenario was further precised in various ways. In [10, 31] it was assumed that robots have unbounded memory, while in [1, 3, 11, 12, 13, 22, 27, 29] robots were oblivious, i.e., it was assumed that they do not have any memory of past observations. Oblivious robots operate in Look-Compute-Move cycles, similar to those described in our scenario. The main differences are in the amount of synchrony assumed in the execution of the cycles. In [1, 3, 31] cycles were executed synchronously in rounds by all active robots, and the adversary could only decide which robots are active in a given cycle. In [10, 11, 12, 13, 21, 22, 27, 29] they were executed asynchronously: the adversary could interleave operations arbitrarily, stop robots during the move, and schedule Look operations of some robots while others were moving. Another difference is in the type of visibility, depending on whether the robots can see the entire space (unlimited visibility) [1, 10, 11, 12, 13, 21, 27, 29, 31] or only within a constant radius (limited visibility) [3, 22].

The model for the discrete scenario introduced in [25] and used in this paper is based on the asynchronous model used in [11, 12, 21, 22, 27, 29]. In particular, it assumes the same characteristics of the robots (anonymity, obliviousness, multiplicity detection) and the same possibilities of the adversary concerning interleaving operations performed by various robots. The only difference is in the execution of Move operations, which has been adapted to the context of graphs: moves of the robots are executed instantaneously from a node to its neighbor, and hence robots always see other robots at nodes. Notice that instantaneous movements are assumed by the semi-synchronous and synchronous models in the continuous scenario (e.g., [1, 3, 31]).

Recently (cf. [20]) we used the discrete scenario introduced in [25] (and also assumed in the present paper) to study the exploration problem in the ring. By contrast with our current situation we showed that the minimum number of robots sufficient to explore a n-node ring is $\Theta(\log n)$. It should be noted that in this very weak model, the exploration problem is significantly more difficult than gathering (considered in [25]). This is due to the fact that in gathering the accomplishment of the task is readily seen in a snapshot: all robots are in one node. By contrast, in order to complete exploration, robots have to "remember" which nodes were visited. Since they do not have any memory of past events, this recollection must be coded in the dynamically changing configurations, and the design of this coding is one of the main challenges of exploration by oblivious robots.

2. Terminology and Preliminaries

We consider a n-node anonymous unoriented tree. Some nodes of the tree are occupied by robots. We will always assume that in an initial configuration of robots there is at most one robot in each node. The number of robots is denoted by k. A *complete d-ary tree* is a rooted tree, all of whose internal nodes have d children and all of whose leaves are at the same distance from the root. Nodes v and w are *similar* if there exists an automorphism of the tree T which carries v to w. A tree is *rigid* if it has no non-trivial automorphisms.

In order to formally define what a robot perceives during a Look action, we introduce the notion of the *view* of a rooted tree T occupied by robots, from its root v. This is defined by induction on the height of the tree T. If T consists only of v then $View(T, v) = (x, \emptyset)$, where $x = 0$, $x = 1$, or $x = *$, if there is 0, 1, or more than 1 robot in v, respectively. If T is of positive height, let $v_1, \ldots, v_m$ be children of the root v, and let $T_1, \ldots, T_m$ be subtrees rooted at $v_1, \ldots, v_m$, respectively . Then $View(T, v) = (x, \{View(T_1, v_1), \ldots, View(T_m, v_m)\})$, where x has the same meaning as before. Now, the snapshot taken by a robot located at v is simply $View(T, v)$. This formalism captures two essential assumptions about the perceptions of robots. First, a robot can distinguish between nodes occupied by 0, 1, or more than 1 robot, but cannot distinguish between numbers larger than 1 of robots located at the same node. Second, subtrees

rooted at children of a node are not ordered: this is captured by considering the set of respective views, and not their sequence, in the recursive definition.

The following notions will be crucial for our considerations.

Definition 1. *Two robots located at nodes v and w are called* equivalent*, if $View(T, v) = View(T, w)$. A node that is not occupied by any robot is called* empty*. When a node is occupied by more than one robot, we say that there is a* tower *in this node. A robot that is not a part of a tower is called* free*.*

An exploration algorithm is a function whose arguments are views, and whose value for any given view $View(T, v)$ is either v or the equivalence class of one of its neighbors, with respect to the following equivalence relation $\sim$: $w_1 \sim w_2$ if there exists an automorphism f of the tree which fixes v, carries empty nodes to empty nodes, free robots to free robots, towers to towers, and such that $f(w_1) = w_2$. Note that $w_1 \sim w_2$ is equivalent to $View(T, w_1) = View(T, w_2)$. If the equivalence class returned by the algorithm for some view has more than one element then the choice of the neighbor in this class to which the robot will actually move, belongs to the adversary. If the value is v, we say that the move of the robot for the given view is the *null move*.

We say that exploration of a n-node tree is possible with k robots, if there exists an algorithm which, starting from any initial configuration of k robots without towers, and for any behavior of the adversary controling asynchrony and choices between equivalent neighbors, explores the entire tree and brings all robots to a configuration in which they all remain idle, i.e., there exists a time t after which all nodes are explored and all subsequent moves of robots are null moves. In fact, our negative results hold even for this weak (implicit) stopping condition, and our positive results (algorithms) are valid even with the following stronger (explicit) stopping condition: for any execution of the algorithm, there exists a time t after which all nodes are explored, and each robot knows that no non-null move of any robot (including itself) will ever occur. Obviously, if $k = n$, the exploration is already accomplished, hence we always assume that $k < n$.

3. Exploration of Trees

In this section we prove that, in general, exploration of n-node trees might require $\Omega(n)$ robots (even if the max degree is 4); we prove that, on the other hand, the minimum number of robots sufficient to explore all n-node trees of maximum degree 3 is $\Theta(\frac{\log n}{\log \log n})$.

3.1. Exploration of Arbitrary Trees

We show that there are arbitrarily large trees of maximum degree 4 whose exploration requires $\Omega(n)$ robots.

Theorem 1. *Exploration of a n-node complete 3-ary tree requires $\Omega(n)$ robots.*

Proof. Let T be a complete 3-ary tree of size n. Consider the following adversary. Two robots act simultaneously if they have the same view. Otherwise, they act sequentially. This means that each robot in turn completely executes its cycle Look-Compute-Move before the next robot proceeds with its own complete cycle. This also implies that whenever two robots form a tower, they will stay together forever. It further means that the number of entities (free robot or tower) is non-increasing.

Consider a group of three leaves having the same parent u. If at least two of these leaves are empty, then the choice of the actual leaf visited by a robot deciding to explore an empty leaf is made by the adversary. Its choice is the following: whenever it is possible, choose an already explored leaf. Thus the algorithm is forced to place an entity in the first two leaves and keep them there until a third entity is able to explore the third leaf. Therefore there exists a time t when these three leaves are occupied. There are only two types of entities: free robots and towers. Thus among the three entities occupying the three leaves under consideration, there are at least two entities of the same type. From this time on, either these entities remain idle forever or they are forced to merge into one, when moving to the parent u (since they have the same view). Hence, when a group of three leaves is explored, the number of entities available to explore the rest

of the tree decreases by at least one. Since a complete 3-ary tree of size $n \geq 4$ has at least $n/6$ groups of three leaves with a common parent, a team of robots succeeding in exploring this tree must contain at least $n/6$ robots. □

3.2. Exploration of Trees of Maximum Degree 3: Upper Bound

Our main result is the following upper bound on the size of the team of robots capable to explore all n-node trees of maximum degree 3.

Theorem 2. *There exists a team of $O(\frac{\log n}{\log \log n})$ robots that can explore all n-node trees of maximum degree 3, starting from any initial configuration.*

This result is proved by showing an exploration algorithm using $O(\frac{\log n}{\log \log n})$ robots.

3.2.1. Overview of Algorithm Tree-exploration

The main idea of the algorithm is the following. The entire tree is partitioned into two or three subtrees, called pieces, the number of parts depending on the shape of the tree. Pieces are explored one after another by a team of three robots that sequentially visit leaves of this piece. Since individual robots do not have memory, a specially constructed, dynamic configuration of robots, called the "brain", keeps track of what has been done so far. More precisely, the brain counts the number of already visited leaves and indicates the next leaf to be visited. It is also the brain that requires most of the robots used in the exploration process. The reason why $\Theta(\log n/\log \log n)$ robots are sufficient for exploration, is that the counting process is efficiently organized. The counting module of the brain consists of disjoint paths of logarithmic lengths, which are appropriately marked by groups of robots of bounded size. Paths are of logarithmic lengths because longer paths cannot be guaranteed to exist in all trees of maximum degree 3. Inside each of these paths a tower moves, indicating a numerical value by its position in the path. The combination of these values yields the current value of the number of visited leaves. Since the number of leaves may be $\Theta(n)$, we need a number x of paths, which can produce $\Theta(n)$ combinations of values, i.e., such that $(\Theta(\log n))^x = \Theta(n)$. This is the reason of constructing $\Theta(\log n/\log \log n)$ paths and thus using $\Theta(\log n/\log \log n)$ robots. We show how to construct these paths in any tree of maximum degree 3, and how to organize the counting process. The latter is complicated by the asynchronous behavior of the robots. During the switch of the counter from value i to $i+1$ robots move in the paths and a snapshot taken during the transition period shows a "blurred" picture: the old value is already destroyed while the new one is not yet created. This could confuse the robots and disorganize the process. Thus we use two counters acting together. They both indicate value i, then one of them keeps this value and the other transits to $i+1$. When this is completed, the first counter transits to $i+1$ and so on. This precaution permits to keep track of the current value during the process of incrementation. During the exploration of one piece of the tree, the brain is located in another piece and controls exploration remotely. After completing the exploration of one piece, the brain is moved to the already explored piece in order to let the exploring agents visit the rest of the tree.

There are two main difficulties in our algorithm. The first is to break symmetries that can exist in configurations of robots, in order to let them act independently and reach appropriate target nodes, in particular during the construction of the brain. The second challenge is the construction and relocation of the brain, as well as organizing its proper functioning by coordinating the two counters, regardless of the behavior of the adversary that controls asynchrony.

The algorithm is divided into the following phases. Phase 1 consists in moving all robots down the tree oriented in a specific way, without creating a tower, in order to create a large zone free of robots. When no robot can move further down, a tower is created to mark the end of Phase 1. In Phase 2, robots are moved from one piece of the tree and create the brain in another piece. If there are local symmetries, a leader is elected and breaks them by relocating to specific nodes of the tree. This is done to let the robots move independently from one piece of the tree to another and occupy target positions. As a consequence, one piece becomes almost empty, which facilitates its exploration. Phase 2 ends when the brain is at its place, properly initialized, and there remain only a tower and a free robot in the other piece, that will explore this piece. Phase 3 is the actual exploration of the piece not containing the brain (or the larger of the two pieces

not containing the brain). This is done by visiting its leaves, one similarity class after another. Inside a similarity class, leaves are explored in a DFS manner, the brain keeping track of the progress of exploration. This phase ends when the brain indicates that the exploring robots are at the last leaf of the explored piece. In Phase 4 the brain is relocated to the already explored piece, and the exploring robots move to one of the unexplored pieces. Again, Phase 4 ends when all robots are in their places and the brain is properly reinitialized (with the indication that one piece is already explored). Finally, in the remaining phases the rest of the tree is explored, similarly as in Phase 3. There is a mechanism in the algorithm that enables robots to see what is the current phase, in order to avoid circular behavior. This is implemented by a special arrangement of robots, called signal, whose value increments from phase to phase.

Summary of Algorithm Tree-exploration

Phase 1: There is no tower in the snapshot.
  Empty the core zones of all pieces by moving down the robots
  {*Let P be the heaviest piece*}
  Create a tower in P outside its core zone
Phase 2: There is at least one tower and no signal in the snapshot.
  {*Let Q be the largest among pieces other than P*}
  Move robots from P in order to construct the brain in Q
  Empty the third piece (other than P and Q), if it exists
  Create the signal, with value 3
Phase 3: The value of signal is 3.
  {*Let P'' be the largest of the pieces other than Q.*}
  Explore P''
Phase 4: The value of signal is 4.
  If there are only two pieces and Q has few leaves
    Explore Q immediately without using a brain
  Else
    Relocate the brain from Q to P''
Phase 5: The value of signal is 5.
  Explore piece Q (as in Phase 3)
  If there are only two pieces
    Stop
  Else
    Increment the value of signal to 6
Phase 6: The value of signal is 6.
  Reinitialize the brain
  Relocate the exploring solitaire to the unexplored piece
  Increment the value of signal to 7
Phase 7: The value of signal is 7.
  Explore the last piece
  Stop

3.2.2. Tools and Basic Properties

Before giving a detailed description of the algorithm we present some concepts that we will use in this description, and prove their basic properties. Let T be a n-node tree of maximum degree 3. Consider a team of k robots, where $c \log n / \log \log n \leq k \leq 2c \log n / \log \log n$, for an appropriately chosen constant c, and $k \equiv 5 \pmod 6$. The conditions on the constant c are explicitly given after the description of the algorithm.

Pieces.

For each internal node v, consider the number of nodes in each of the subtrees rooted at neighbors of v, and let n_v be the maximum of these numbers. It is well known that either there exists exactly one node v

for which $n_v \leq (n-1)/2$ (the centroid), or there is exactly one edge $\{v, w\}$, for which $n_v = n_w = n/2$ (the bicentroid). In each case we consider the oriented tree from the centroid or bicentroid down to the leaves. We will say that the tree is rooted in the centroid or bicentroid and use the usual notions of parent and children of a node.

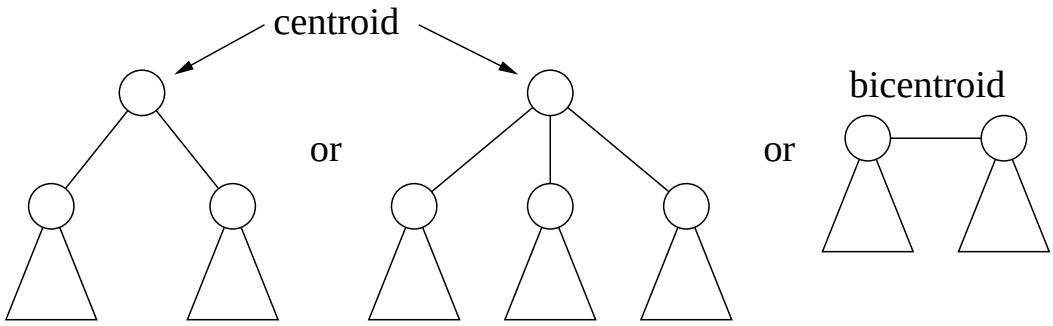

Figure 1: Possible cases of the tree structure: two or three pieces

Next we define the subtrees of T, called its *pieces*. If T has a centroid of degree 2 then there are two pieces T_1 and T_2 which are rooted at children of the centroid. If T has a centroid of degree 3 then there are three pieces T_1, T_2 and T_3 which are rooted at children of the centroid. Finally, if T has a bicentroid then there are two pieces T_1 and T_2 which are rooted at nodes of the bicentroid (see Figure 1). Without loss of generality we assume that sizes of T_1, T_2 and T_3 are non-increasing. Hence $(n-1)/3 \leq |T_1| \leq n/2$ and $n/4 \leq |T_2| \leq n/2$. For every piece, we define its *weight* as the number of robots located in it. Thus we talk about the heaviest piece, a heavier piece, etc. A piece T_i is called *unique* if there is no other piece whose root has the same view as the root of T_i.

Core Zone.

A node in a piece is *a core node*, if the size of the subtree rooted at this node is larger than the size k of the entire team of robots. The set of core nodes in a piece is called the *core zone* of the piece.

Lemma 1. *In any rooted tree of size x and such that every internal node has at most two children, the size of the core zone is at least $\frac{x+1}{k+1} - 1$.*

PROOF. Let $f(x)$ denote the minimum size of the core zone in a tree of size x. Thus we have $f(x) = 0$, when $x \leq k$, and

$$f(x) = 1 + \min_{x_1+x_2=x-1}(f(x_1) + f(x_2)) \ ,$$

when $x > k$. We prove the lemma by induction on x. It is true for $x = k$. Suppose it holds for $k \leq x' < x$. We prove it for x.

Suppose that $x_1 \leq k$ or $x_2 \leq k$. Without loss of generality assume $x_1 \leq k$. We have

$$f(x) = 1 + f(x_2) \geq 1 + f(x-k-1) \geq 1 + \frac{x-k}{k+1} - 1 = \frac{x+1}{k+1} - 1 \ .$$

If $x_1, x_2 > k$, we have

$$f(x) = 1 + \min_{x_1+x_2=x-1}(f(x_1) + f(x_2)) \geq 1 + \min_{x_1+x_2=x-1}((\frac{x_1+1}{k+1} - 1) + (\frac{x_2+1}{k+1} - 1))$$

$$\geq 1 + \frac{(x-1)+2}{k+1} - 2 \geq \frac{x+1}{k+1} - 1 \ .$$

□

Since the size of any of the two largest pieces is at least $n/4$, Lemma 1 implies that the size of the core zone of any of these pieces is at least $\frac{n \log\log n}{10c \log n}$.

Descending Paths.

The basic component of the brain is a *descending path*. This is a simple path in a piece Q, whose one extremity is its node closest to the root of Q. It will be called the beginning of the path. The other extremity will be called its end, cf. Figure 2. The size of such a path is the number of its nodes. We need sufficiently many pairwise disjoint descending paths, each sufficiently long, for all parts of the brain. The construction is a part of the proof of the following lemma.

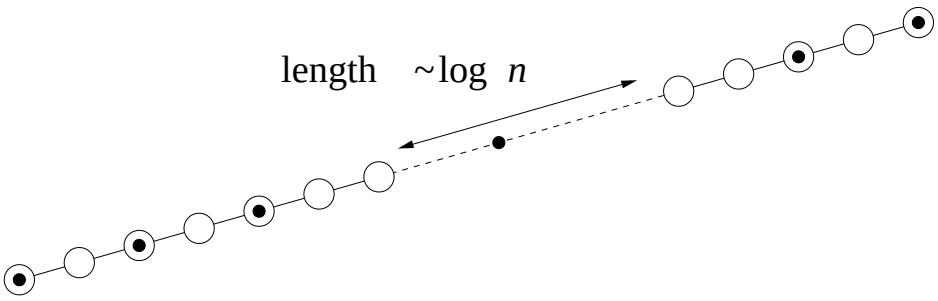

Figure 2: Descending path with its extremities

Lemma 2. *For any sufficiently large m, every tree of maximum degree 3 and of size m contains at least $\log^2 m$ pairwise disjoint descending paths of size at least $\frac{1}{4}\log m$.*

Proof. Let m be an integer. Consider a rooted tree U of maximum degree 3 and of size m. It contains a descending path of size at least $\log m - 2$. Take such a path p_1 of size $\lfloor \log m - 2 \rfloor$ with the beginning at the root of U. In $U \setminus p_1$ there are at most $\log m$ pairwise disjoint subtrees whose union has size at least $m - \log m$. One of these trees must be of size at least $(m - \log m)/\log m$. It must contain a descending path of size at least $\log((m - \log m)/\log m)$. Take such a path p_2 of size $\lfloor \log((m - \log m)/\log m) \rfloor$. In $U \setminus \{p_1 \cup p_2\}$ there are at most $2\log m$ pairwise disjoint subtrees whose union has size at least $m - 2\log m$. One of these trees must be of size at least $(m - 2\log m)/(2\log m)$. Continuing in this way, we construct $s = \lceil \log^2 m \rceil$ pairwise disjoint descending paths of size at least

$$\log \frac{m - s\log m}{s \log m} \geq \frac{1}{2}\log m - \log s \geq \frac{1}{2}\log m - 4\log\log m \geq \frac{1}{4}\log m \ ,$$

for m sufficiently large. □

The core zone is a tree of maximum degree 3, rooted in the root of the piece and has size $m \geq \frac{n \log\log n}{10c \log n}$. Hence, for sufficiently large n, Lemma 2 guarantees the existence of at least $\log^2 m \geq \log n \geq 5\log n/\log\log n$ pairwise disjoint descending paths of size at least $\frac{1}{4}\log m \geq \frac{1}{8}\log n$ in any of the two largest pieces. These $\Theta(\log n/\log\log n)$ descending paths of size $\Theta(\log n)$ are used to construct the brain.

Modules of the Brain.

The brain consists of four parts: two *counters*, the *semaphore* and the *garbage*. Descending paths forming these parts will be situated in the core zone of a piece, each of the paths at distance at least 3 from the others, in order to allow correct pairing of beginnings and ends.

We now describe the structure of a counter. This is a collection of $q \in \Theta(\frac{\log n}{\log\log n})$ pairwise disjoint descending paths, of sizes $L+1, L+2, \ldots L+q$, where $L \in \Theta(\log n)$. We take paths of different lengths in order to easily distinguish them. Nodes of the ith path are numbered 1 to $L+i$ (where 1 corresponds to the beginning). Two towers will be placed in the first and third nodes of each path, thus marking its beginning. Similarly, three towers will be placed at the end of each path, separated by empty nodes, thus marking its end. Moreover there will be two or three robots moving from node 7 to node $L-8$ of each path.

If these robots are located in the same node (thus forming a tower), their position codes a numerical value. By combining these values on all paths, we obtain the value of the counter. Since on each path there are $L - 14$ available positions, the value of the counter is the resulting integer written in base $L - 14$.

Let $q = \lceil 2 \log n / \log \log n \rceil$ and $L_1 = \frac{1}{10} \log n$. Take q of the descending paths described in the proof of Lemma 2 (chosen in an arbitrary deterministic way, identical for all robots, and excluding p_1), and in the ith path, where $1 \leq i \leq q$, take the lower part of size $L_1 + i$. These will be the descending paths of the first counter. Similarly, let $L_2 = L_1 + q + 1$. Take a set of q descending paths, other than those used for the first counter and other than p_1. In the ith path, where $1 \leq i \leq q$, take the lower part of size $L_2 + i$. These will be the descending paths of the second counter.

Another module of the brain is the semaphore consisting of two of the descending paths constructed in the proof of Lemma 2 (again excluding p_1). In each of these paths take the lower part of distinct constant sizes. The beginning and end of each path is marked similarly as in the counter. Likewise, there are two or three robots moving in each of these paths, their possible locations restricted to node 7 and 8 in each path. In each path, if these robots are located in the same node (thus forming a tower), they code one bit. Thus the semaphore has 4 possible values $00, 01, 11, 10$.

Finally, the garbage is the first descending path p_1 constructed in the proof of Lemma 2. This path has the property that its beginning is at the root of the piece. This path has length at least $\frac{1}{8} \log n$, and thus larger than the total number of robots, for sufficiently large n. The role of the garbage is to store all robots of the brain not used for the counters and the semaphore. The garbage is filled by putting a tower or a robot every 5 nodes in the path, until all robots are disposed. Therefore the end of the path is marked similarly as for paths in the counter and the semaphore, but the beginning is left unmarked.

Ordering of Robots.

We first define a total order $\sqsubset$ on the set of all views. Let $V = View(T, v)$ and $V' = View(T', v')$. If the height of T is smaller than the height of T' then $V \sqsubset V'$. Otherwise, if the height of both trees is 0 then $(x, \emptyset) \sqsubset (x', \emptyset)$, if $x \leq x'$, where $0 < 1 < *$. If the height of both trees is positive, the order of views is the lexicographic order on the sequences $(x, View(T_1, v_1), \ldots, View(T_m, v_m))$, where views at children are ordered increasingly by induction.

We now define the following total preorder $\leq$ on the robots in the rooted tree T. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two robots located at nodes v_1, v_2, at distances d_1 and d_2 from the root. (In the case of the bicentroid, we consider the distance to its closer extremity.) We let $\mathcal{R}_1 \leq \mathcal{R}_2$, if and only if, $d_1 < d_2$, or $d_1 = d_2 \wedge View(T, v_1) \sqsubset View(T, v_2)$. Note that the equivalence relation induced by this preorder is exactly the equivalence between robots defined previously. We say that a robot is larger (smaller) than another one meaning the preorder $\leq$. A robot not equivalent to any other is called *solitaire*.

Lemma 3. *The number of equivalent free robots in any piece is either 1 or even.*

Proof. Consider an equivalence class of robots restricted to a piece. Call it S. Assume that $|S| \geq 2$. Let v be the least common ancestor of all locations of robots in S. Since equivalent robots are at the same distance from the root, v must have two children. Let T_1 and T_2 be subtrees rooted at these children. Since views of all robots in S are identical, the number of robots of S in T_1 is equal to the number of robots of S in T_2. Hence S is of even size. □

It follows from Lemma 3 that any unique piece with an odd weight and no towers of more than two robots must contain a solitaire.

3.2.3. Description of Algorithm Tree-exploration

Phase 1. There is no tower in the snapshot.

Goal: Empty the core zones of all pieces and create one tower in one piece.

We first free the core zones by moving every robot to an empty child, as long as such a child exists, except for up to two robots that may move from one piece to another. As described below, these exceptional robots are solitaires. The objective here is to have a unique heaviest piece with the additional property that it is either of odd weight or completely occupied by robots (i.e. every node of the piece is occupied

by a robot). This is always possible because $k \equiv 5 \pmod 6$. Indeed, if there are two heaviest pieces, then there must exist a third piece of odd weight, and thus a solitaire of this piece (whose existence is guaranteed by Lemma 3) can move to one of the heaviest pieces, thus breaking the tie. If there is a unique heaviest piece, but of even weight and not completely occupied by robots, then there must exist another piece of odd weight, and thus a solitaire of this piece (whose existence, again, is guaranteed by Lemma 3) can move to the heaviest piece. Note that the case of three heaviest pieces is impossible because k is not divisible by 3. In the two cases where a solitaire has to go from one piece to another, this solitaire is chosen as the smallest one. This way, it can move to another piece without passing through a node occupied by other robots. (Indeed, if its parent node were occupied by a robot, this robot would also be a solitaire, smaller than the given one.)[3]

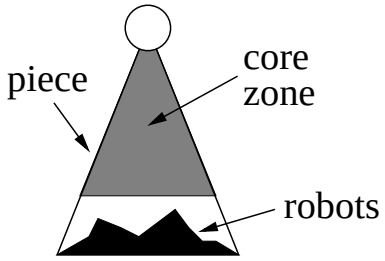

Figure 3: The core zone and the position of the robots at the end of Phase 1 just before creating the tower

As soon as the required properties hold in a piece P and the core zones are empty (except for possibly one robot in the core zone of P), see Figure 3, a tower is created outside the core zone of P by moving a solitaire to an occupied node in such a way that at least half the robots in P, including a solitaire, are located outside the subtree rooted at the tower. The latter precaution is taken to have enough robots to form and subsequently move towers in Phase 2 using the solitaire. The way this is done will be described in the sequel.

Phase 1 has been clearly identified by the absence of towers in the snapshot. Such an easy characterization is not available in the subsequent phases, hence we use a gadget called *signal* to identify them. A signal is a largest set of at least 4 towers situated on a descending path inside a piece, such that consecutive towers are separated by two empty nodes. The value of a signal is $x - 1$, where x is the number of towers in it. This value will indicate the number of the current phase.

Phase 2. There is at least one tower and no signal in the snapshot.

In this phase piece P can be recognized as the unique piece where there is a tower outside the core zone and Q as the largest among pieces other than P (in the case of a tie Q is any piece with robots in the core zone.) Notice that, at the beginning of Phase 2, the core zone of Q does not contain any robots. Hence there is room in it for the brain.

Goal: Construct and initialize the brain in the core zone of piece Q, prepare the other pieces for exploration, and create the signal.

Stage 1. Goal: Move robots from P in order to construct the brain in Q and prepare P for exploration.

We now describe the way to form towers in P and move them to appropriate places in the descending paths forming the brain in Q. Robots migrate from piece P to piece Q, one or two robots at a time. The next robot or pair of robots starts its trip from P to Q only after the previous one is at its place. The aim is to occupy target nodes by towers. Nodes in descending paths are filled one path after another in a DFS post-order of beginnings of the paths. Thus a tower occupies a node v only after all robots in the

[3]In the sequel, we skip the technical details of how a solitaire is chosen when it is needed. In each case the choice is not difficult and usually more than one solitaire could do the job.

subtree rooted at v are in their target positions. This rule applies to all descending paths of the brain, except the garbage. The latter is constructed at the end, after all other parts of the brain are completed. This is possible because the descending path containing the garbage starts at the root of the piece (path p_1 described in the proof of Lemma 2). The above migration of towers is done until there remains only a single tower and a solitaire in P. This prepares P for exploration.

There are two difficulties in performing this migration, both due to symmetries in configurations of robots. The first difficulty is to form towers consisting of only two robots in P and the other is to place such a tower in a specific target node in Q. (We want to restrict the size of towers in order to be able to create many of them using the available robots).

The essence of the first difficulty is that equivalence classes of robots can be large and thus it may be difficult to form a single small tower. (For example, if all robots in a piece are equivalent and occupy the same level, a single small tower cannot be formed without outside help.) We solve this problem by using a solitaire to break symmetry between two equivalent robots. More precisely, the solitaire moves to meet one of the equivalent robots thus creating a tower of two robots. At the same time the other equivalent robot becomes a solitaire.

The essence of the second difficulty is that if there are at least two equivalent target positions that a tower could occupy, the adversary could break the tower at the time when the tower tries to go down from the least common ancestor of these target nodes, sending each of the robots forming the tower to a different target node. We solve this problem by using a solitaire to first break the symmetry between these target positions. This solitaire, called the *guide* of the tower, is placed in one of these positions, thus indicating that the tower should go to the closest of the equivalent positions. As soon as the tower reaches its target, the solitaire is again available to break other symmetries, either those encountered when forming towers in P or when placing them in Q.

Stage 1 ends with the brain constructed in the core zone of piece Q. Moreover, in piece P there remain only a single tower and a robot without towers in its ancestors.

Stage 2. Goal: Empty the third piece P' (other than P and Q), if it exists.

This is done as follows. A largest robot of P', not in the root of P' (either a free robot or in a tower) goes to its parent. When there are no robots outside the root, the robots from the root of P' go to the garbage in Q. This way of merging all robots of P' at the root of this piece prevents accidental creation of a signal. Stage 2 ends when the ending condition of Stage 1 holds and piece P', if it exists, is empty.

Stage 3. Goal: Create the signal.

The signal is created at the bottom of the garbage (without considering towers marking its end). Towers descend in the garbage one at a time, until two sequences consisting of 4 towers, each at distance 3 from the preceding one, are created. These two sequences are separated by 5 empty nodes. Since there is no longer sequence of this type in the entire tree, the value of the newly created signal is 3. This completes Stage 3 and the entire Phase 2. (Note that we use two sequences forming a signal, rather than just one, in order to be able to move one of these sequences later on, without destroying the value of the signal. In fact we also need to leave two additional towers between these sequences, in order to update the value of signal from 3 to 4, when passing to Phase 4.)

From now on all towers in the entire tree are separated by at least one empty node. Hence if a tower moves and the adversary breaks it by holding back some of the robots of the tower, this can be recognized in subsequent snapshots and the moving tower can be reconstructed. Note that from now on we need not specify the existence of a tower in the snapshot, since the signal contains towers.

Phase 3. The value of signal is 3.
Goal: Explore P'': the largest of the pieces other than Q. (We explore this piece first to be able to relocate the brain into it in Phase 4: the other piece could be too small.)

At the beginning of Phase 3 both counters indicate value 0. Piece P'' will be explored by the free robot and the tower that are currently outside Q. They will be called exploring robot and exploring tower, respectively. These two entities explore leaves of P'' one similarity class after another in increasing order, induced by any total preorder of the nodes, with the following property: the equivalence classes induced by

this preorder are the previously defined similarity classes. The entities move only if both counters indicate the same value i. Suppose that the jth class has size s_j. Let r be such that $i = s_1 + \cdots + s_d + r$, with $r \leq s_{d+1}$. Hence the brain indicates that the next leaf to be explored is the rth leaf in the $(d+1)$st class. If $r = 1$, the exploring robot goes to any leaf of the $(d+1)$st class. Otherwise, consider two cases. If r is even then let u be the leaf where the exploring robot is located. In this case the exploring tower goes to the (unique) closest leaf in the same similarity class. If r is odd then let v be the leaf where the exploring tower is located. In this case the exploring robot goes to the leaf w determined as follows. Let j be the length of the longest sequence of 1's counted from the right (least significant bit) of the binary expansion of the integer $(r-3)/2$. Order all leaves of the similarity class of v in any non-decreasing order of distances from v. The leaf w is the 2^{j+1}th node in this order. Notice that w is a closest leaf from v not yet explored.

Incrementing values of both counters from i to $i+1$ and moving the exploring robots according to those increments are complex actions involving relocation of many robots. Due to asynchrony, snapshots can be taken during these complex actions, potentially disorganizing the process. To ensure correct exploration, we artificially synchronize these actions using the semaphore. Its values change in the cycle $00, 01, 11, 10, 00, \ldots$. Note that the changes of values of the semaphore do not need additional synchronization, as each change involves a move of only one robot or tower. In the case of a move of a tower, the adversary can split the tower by delaying some of its robots and moving others, hence the value of the corresponding bit is unclear and robots must decide which value should be set. Nevertheless this is never ambiguous: for example, if the value of the first bit is 0 and the second is unclear, it must be set to 1 because, when the first bit is 0, the only possible change of the second bit is from 0 to 1. Other cases are similar.

At the beginning of Phase 3 the semaphore is at 00. This indicates that the first counter has to modify its value to $i+1$, where i is the current value of the second counter. When this is done, the value of the semaphore changes to 01. This indicates that the second counter has to modify its value to the current value of the first counter. When this is done, the value of the semaphore changes to 11. This indicates that the exploring robot or the exploring tower (depending on the parity of the value shown by the counters) has to move to the neighbor of the leaf it occupies. When this is done, the value of the semaphore changes to 10. This indicates that the exploring entity which is in an internal node (i.e., the one that has just moved) has to move to the leaf indicated by the value of both counters, as explained above. When this is done, the value of the semaphore changes to 00.

Phase 3 is completed when the semaphore has value 11 and both counters have value $f+1$, where f is the number of leaves in piece P''. At this time the value of signal is changed from 3 to 4 (by moving an additional tower down the garbage), thus marking the end of this phase. Note that, when both counters have value $f+1$, all leaves of P'' are explored. There are two cases. If $P'' = P$ then at least one path between the root and a leaf of P'' has been explored when P was evacuated. Otherwise, at least one path between the root and a leaf of P'' has been explored when the exploring solitaire came from P to explore P''. Hence in both cases all leaves and at least one path between the root and a leaf have been explored. Since by the description of the exploration the explored part of P'' is connected, this implies that the entire piece has been explored.

Phase 4. The value of signal is 4.
Goal: Relocate the brain from Q to P'' (except when there are only two pieces and Q has few leaves, in which case exploration of Q is done immediately: see Subcase 2.2).

While the brain is relocated to P'', piece Q is emptied and thus ready to be explored. Piece Q is emptied in reverse order of its filling in Phase 2, i.e., robots that came last to Q leave it first. We will need the exploring solitaire and tower in piece Q in order to perform exploration during Phase 5. Hence while towers forming the old brain move from Q to P'', the solitaire and the exploring tower move in the opposite direction. This creates a problem when the tree has a long path of nodes of degree two, between the old brain and piece P'': there is no room to cross on this path. Hence for this class of trees we will use a particular technique. Consider two cases.

Case 1. There exist nodes v and w outside P'' such that the path from the root of P'' to each of them does not contain robots and there exists a path from a tower in Q to the root of P'' not containing robots and

not containing v or w.

In this case there is no crossing problem. The solitaire and the exploring tower from Q can hide in v and w to let the towers from Q (that formed the old brain) move to P''.

Case 2. There are no nodes v and w as described in Case 1.

Let M be the largest integer such that $10c \log M / \log \log M \geq \log M$.

– Subcase 2.1. The number f of leaves in piece Q is larger than M.

Since any tree of maximum degree 3 containing f leaves has height at least $\log f$, the condition on integer M implies that there exists a descending path in Q, with beginning u, satisfying the following properties:

(1) it is able to store all towers needed to explore Q (i.e. $2c \log f / \log \log f$ of them), leaving distance 4 between consecutive towers. (We leave distance 4 not to confuse the sequence of towers with a signal.)

(2) there exist two leaves outside the tree rooted at u.

All towers from Q are moved to the above descending path leaving 3 empty nodes between consecutive towers, with the following exception. When moving the first five towers, the value of the signal is recreated using these towers. This is done before moving the second sequence of the signal created in Phase 2. After moving $2c \log f / \log \log f$ towers, all additional towers from this path are collapsed to one tower. After this compacting the condition of Case 1 holds because of property (2).

– Subcase 2.2. The number of leaves in piece Q is at most M.

In this case there are so few leaves that we can explore all of them without using a brain. First we recreate the signal in P with value 4, to record the phase number. Then all robots from Q go to the leaves. When all leaves are occupied, all robots go towards the root of Q forming a tower in this root, thus exploring the remaining nodes of Q. At this point the algorithm stops (explicit stopping condition).

Thus, after a finite number of moves in Case 2, either the exploration is completed (Subcase 2.2) or the algorithm transits to Case 1. From now on we suppose that the condition of Case 1 holds.

We continue Phase 4 by creating a signal with value 4 in piece P''. This is done by moving towers from the top of the garbage in Q and placing them outside the core zone in P''. The path forming the signal is of bounded length and thus there is enough space outside the core zone to place it. Moreover we place three additional towers in this path to be able to subsequently increase the value of the signal up to 7. After this is done we create the new brain in P'', similarly as in Phase 2. In particular, we use the solitaire as a guide to direct the towers coming from Q to their target positions. Note that all towers and robots in the core zone of Q are alone in their equivalence classes and thus there is no need to break symmetries using solitaires. When the counters and the semaphore of the new brain are created in P'', all robots from Q, except the exploring tower and solitaire are moved to the garbage of the new brain. Note that all the above actions are possible, since the solitaire and towers are able to move between pieces Q and P'' without crossing problems.

When there is only the exploring tower and solitaire in Q, the value of signal in P'' is incremented to 5. This ends Phase 4.

Phase 5. The value of signal is 5.

Goal: Explore piece Q and stop if there are only two pieces.

We proceed exactly as in Phase 3, this time exploring piece Q instead of P''. When the brain indicates that all leaves are explored, two situations are possible. If there are only two pieces in the tree, all nodes are already explored and the algorithm stops (explicit stopping condition). If there are three pieces, the value of signal is incremented to 6. This ends Phase 5.

Phase 6. The value of signal is 6.

Goal: Reinitialize the brain and relocate the exploring solitaire to the unexplored piece.

Both counters in the brain are reset to 0, the semaphore is reset to 00. The exploring solitaire moves to the root of the unexplored piece (the only piece without towers). The value of signal is incremented to 7. This ends Phase 6.

Phase 7. The value of signal is 7.
Goal: Explore the last piece and stop.

The piece containing only a solitaire is explored (using this solitaire and the tower from Q). This is done again as in Phase 3. When the brain indicates that all leaves are explored, exploration is completed and the algorithm stops (explicit stopping condition).

It remains to give the conditions on the constant c such that the number k of robots satisfies the inequalities $c \log n / \log\log n \leq k \leq 2c \log n / \log\log n$. It is enough to choose the constant c so that there are sufficiently many robots to form the brain (including the markers of descending paths' extremities) and the exploring team. Note that if there are three pieces in the tree, and robots are initially equally divided among them, only $k/3$ robots will be used.

3.3. Exploration of trees of maximum degree 3 : lower bound

We now prove a lower bound on the number of robots necessary for exploration of complete binary trees, that matches the upper bound given by Algorithm Tree-exploration.

Theorem 3. $\Omega(\frac{\log n}{\log\log n})$ *robots are required to explore n-node complete binary trees.*

Proof. We consider a n-node complete binary tree of height h. Initially, k nodes of the tree are occupied by robots and there is at most one robot in each node. During the exploration robots move, and at any time they occupy some nodes of the tree, forming a *configuration*. The configuration corresponding to the tree of height 0 whose unique node hosts k robots is denoted by the pair (k, ϵ), where ϵ is the empty list. Assume now that $h > 0$. Let i be the number of robots occupying the root. Then the corresponding configuration is inductively defined as $(i, (C_l, C_r))$, where C_l, respectively C_r, is the configuration corresponding to the left, respectively right, subtree of the root. For a given configuration we define its height as h and its number of robots as k.

We define by induction an equivalence relation $\equiv$ on configurations. Only configurations of the same height can be equivalent. First we have $(k, \epsilon) \equiv (k', \epsilon)$ if and only if $k = k'$. We now define equivalence of configurations of positive height. Consider two configurations $(i, (C_l, C_r))$ and $(i', (C'_l, C'_r))$ of the same positive height. Then these two configurations are equivalent if and only if $(i = i') \wedge \big((C_l \equiv C'_l \wedge C_r \equiv C'_r) \vee (C_l \equiv C'_r \wedge C_r \equiv C'_l)\big)$. Intuitively, two configurations are equivalent if one can be obtained from the other by switching the left and the right subtrees at some subset of the internal nodes of the tree. Clearly, two equivalent configurations are undistinguishable by the robots.

For every integer h, we define a total order $\preccurlyeq$ on the set of configurations of height h as follows. For any k and k', we have $(k, \epsilon) \preccurlyeq (k', \epsilon)$ if and only if $k \leq k'$. Let $C = (i, (C_l, C_r))$ and $C' = (i', (C'_l, C'_r))$ be two configurations of height $h \geq 1$. Let k and k' be their respective number of robots. Then $C \preccurlyeq C'$ if and only if $(k < k') \vee (k = k' \wedge i < i') \vee (k = k' \wedge i = i' \wedge C_l \preccurlyeq C'_l \wedge C_l \neq C'_l) \vee (k = k' \wedge i = i' \wedge C_l = C'_l \wedge C_r \preccurlyeq C'_r)$.

Using this order, we define the following transformation φ on configurations. We also define φ by induction on the height of the configuration. For any k, we have $\varphi((k, \epsilon)) = (k, \epsilon)$. Let $C = (i, (C_l, C_r))$ be an arbitrary configuration of height $h \geq 1$. If $\varphi(C_l) \succcurlyeq \varphi(C_r)$, then we set $\varphi(C) = (i, (\varphi(C_l), \varphi(C_r)))$. Otherwise, we set $\varphi(C) = (i, (\varphi(C_r), \varphi(C_l)))$. Clearly, C is equivalent to $\varphi(C)$. Notice that we have $\varphi(C) = \varphi(C')$ if and only if $C \equiv C'$. Hence a configuration $\varphi(C)$ is a canonical representative of the equivalence class of C, for any configuration C.

Let $v_{k,h}$ be the size of the set $\{\varphi(C) | C \text{ is a configuration of } k \text{ robots and of height } h\}$.

Claim 1. *For any $k \geq 1$ and $h \geq 1$, we have $v_{k,h} \leq \frac{(h+2)^{2k-1}}{2k-1}$.*

We prove the claim by induction on k. First note that for any integers $h \geq 0$ and $k \geq 0$, we have $v_{0,h} = 1$ and $v_{k,0} = 1$. Fix positive integers h and k. If k is odd, then we have

$$
\begin{aligned}
v_{k,h} = \quad & v_{k,0} \cdot v_{0,h-1} \cdot v_{0,h-1} \\
& + v_{k-1,0} \cdot v_{1,h-1} \cdot v_{0,h-1} \\
& + v_{k-2,0} \cdot \left(v_{2,h-1} \cdot v_{0,h-1} + \frac{v_{1,h-1} \cdot (v_{1,h-1}+1)}{2} \right) \\
& + v_{k-3,0} \cdot (v_{3,h-1} \cdot v_{0,h-1} + v_{2,h-1} \cdot v_{1,h-1}) \\
& + \cdots \\
& + v_{0,0} \cdot \left(v_{k,h-1} \cdot v_{0,h-1} + v_{k-1,h-1} \cdot v_{1,h-1} + \cdots + v_{\frac{k+1}{2},h-1} \cdot v_{\frac{k-1}{2},h-1} \right) \\
= \quad & 1 \\
& + v_{1,h-1} \\
& + \left(v_{2,h-1} + \frac{v_{1,h-1} \cdot (v_{1,h-1}+1)}{2} \right) \\
& + (v_{3,h-1} + v_{2,h-1} \cdot v_{1,h-1}) \\
& + \cdots \\
& + \left(v_{k,h-1} + v_{k-1,h-1} \cdot v_{1,h-1} + \cdots + v_{\frac{k+1}{2},h-1} \cdot v_{\frac{k-1}{2},h-1} \right) \quad .
\end{aligned}
$$

If k is even, then we have

$$
\begin{aligned}
v_{k,h} = & v_{k,0} \cdot v_{0,h-1} \cdot v_{0,h-1} \\
& + v_{k-1,0} \cdot v_{1,h-1} \cdot v_{0,h-1} \\
& + v_{k-2,0} \cdot \left(v_{2,h-1} \cdot v_{0,h-1} + \frac{v_{1,h-1} \cdot (v_{1,h-1}+1)}{2} \right) \\
& + v_{k-3,0} \cdot (v_{3,h-1} \cdot v_{0,h-1} + v_{2,h-1} \cdot v_{1,h-1}) \\
& + \cdots \\
& + v_{0,0} \cdot \left(v_{k,h-1} \cdot v_{0,h-1} + v_{k-1,h-1} \cdot v_{1,h-1} + \cdots + \frac{v_{\frac{k}{2},h-1} \cdot (v_{\frac{k}{2},h-1}+1)}{2} \right) \\
= \quad & 1 \\
& + v_{1,h-1} \\
& + \left(v_{2,h-1} + \frac{v_{1,h-1} \cdot (v_{1,h-1}+1)}{2} \right) \\
& + (v_{3,h-1} + v_{2,h-1} \cdot v_{1,h-1}) \\
& + \cdots \\
& + \left(v_{k,h-1} + v_{k-1,h-1} \cdot v_{1,h-1} + \cdots + \frac{v_{\frac{k}{2},h-1} \cdot (v_{\frac{k}{2},h-1}+1)}{2} \right) \quad .
\end{aligned}
$$

Both above equalities follow from counting possible ways of distributing k robots in the root and in the right and left subtrees, respecting the canonical form of the configurations.

For $k = 1$ and $h \geq 1$, we have $v_{1,h} = 1 + v_{1,h-1}$ and thus $v_{1,h} = h + 1$. Hence the claim holds for $k = 1$ and arbitrary $h \geq 1$.

For $k = 2$ and $h \geq 1$, we have $v_{2,h} = 1 + v_{2,h-1} + \frac{h(h+1)}{2}$. From $h \geq 1$, we get $v_{2,h} \leq v_{2,h-1} + (h+1)^2$. This implies $v_{2,h} \leq 1 + \sum_{i=1}^{h} (i+1)^2 \leq 1 + \int_1^{h+1} (x+1)^2 dx \leq \frac{1}{3}(h+2)^3$. Thus the claim also holds for $k = 2$ and arbitrary $h \geq 1$.

For $h = 1$ and $k \geq 1$, we have $v_{k,1} \leq v_{k-1,1} + \frac{k}{2} + 1$. This implies $v_{k,1} \leq (k+1)(k+2)/2 \leq \frac{1}{2k-1}3^{2k-1}$. Thus the claim holds for $h = 1$ and arbitrary $k \geq 1$.

Fix $k \geq 3$. Assume that the claim holds for any pair (k', h) such that $1 \leq k' \leq k-1$ and $h \geq 1$. Fix $h \geq 2$. Applying the induction hypothesis in the recurrence formula, we obtain (for k either odd or even)

$$\begin{aligned} v_{k,h} \leq \quad & 1 \\ & + (h+1) \\ & + \left(\frac{1}{3}(h+1)^3 + (h+1)^2\right) \\ & + \left(\frac{1}{5}(h+1)^5 + \frac{(h+1)^{3+1}}{3 \cdot 1}\right) \\ & + \cdots \\ & + \left(v_{k,h-1} + \frac{(h+1)^{(2k-3)+1}}{(2k-3)\cdot 1} + \frac{(h+1)^{(2k-5)+3}}{(2k-5)\cdot 3} + \cdots + \frac{(h+1)^{(2\lceil \frac{k}{2} \rceil - 1)+(2\lfloor \frac{k}{2} \rfloor - 1)}}{(2\lceil \frac{k}{2} \rceil - 1)\cdot(2\lfloor \frac{k}{2} \rfloor - 1)}\right) \quad . \end{aligned}$$

For any $1 \leq i \leq \frac{k}{2}$, we have $\frac{1}{(2(k-i)-1)\cdot(2i-1)} \leq \frac{1}{2k-3}$. Thus

$$\sum_{i=1}^{k/2} \frac{1}{(2(k-i)-1)\cdot(2i-1)} \leq \frac{k/2}{2k-3}$$

which in turn is less or equal to $\frac{1}{2}$ for $k \geq 3$. This implies

$$\begin{aligned} v_{k,h} &\leq 1 + (h+1) + \left(\frac{1}{3}(h+1)^3 + (h+1)^2\right) + \left(\frac{1}{5}(h+1)^5 + \frac{1}{2}(h+1)^4\right) + \cdots \\ &\quad + \left(v_{k,h-1} + \frac{1}{2}(h+1)^{2k-2}\right) \\ &\leq 1 + (h+1) + (h+1)^2 + (h+1)^3 + \cdots + (h+1)^{2k-3} + \left(v_{k,h-1} + \frac{1}{2}(h+1)^{2k-2}\right) \quad . \end{aligned}$$

In view of $h \geq 2$, we get $\sum_{i=0}^{2k-3}(h+1)^i \frac{(h+1)^{2k-2}-1}{h} \leq \frac{(h+1)^{2k-2}}{2}$. Therefore

$$v_{k,h} \leq v_{k,h-1} + (h+1)^{2k-2} \quad .$$

This implies

$$v_{k,h} \leq v_{k,1} + \sum_{i=2}^{h}(i+1)^{2k-2} \leq \frac{3^{2k-1}}{2k-1} + \int_2^{h+1}(x+1)^{2k-2}dx \leq \frac{(h+2)^{2k-1}}{2k-1} \quad .$$

This concludes the proof of the claim.

Let us consider a complete binary tree of height h, and suppose that a team of k robots is able to explore it. As an adversary, we decide that all robots act synchronously, i.e., they all look at the same time, compute at the same time and finally move at the same time. Hence, for any configuration, the notion of the next configuration is well defined, and thus a specific sequence S of configurations, starting from the initial one, is defined. Using the equivalence relation $\equiv$, a corresponding sequence S' of classes of configurations is defined. After completing exploration, the robots have to remain idle. Consider the first configuration in the sequence S such that all robots decide not to move after performing the Look operation in this configuration. For the purpose of contradiction, assume that this particular configuration is not among the first $v_{k,h}$ configurations in S. Then, in the sequence S', some equivalence class appears twice, say in positions i and $j > i$. Notice that, if two configurations C_1 and C_2 are equivalent then the next

configuration after C_1 is equivalent to the next configuration after C_2. This enables us to properly define the next class of configurations for a given class. Hence, after the j-th step of the synchronous execution, the sequence S' becomes periodic. Thus, it is impossible that the robots enter a configuration in which all of them decides to remain idle. This is a contradiction. Hence, the robots stop after at most $v_{k,h}$ steps. During each synchronous step, at most k new nodes are explored. Hence, at most $k \cdot v_{k,h} \leq (h+2)^{2k}$ nodes are explored by the robots before they stop. Since all nodes must be explored, this implies $(h+2)^{2k} \geq n$. Since $n = 2^{h+1} - 1$, we can conclude that $\Omega(\frac{\log n}{\log \log n})$ robots are needed to explore the complete binary tree of size n, regardless of the initial configuration. □

4. Exploration of rigid trees

We finally consider exploration of trees that do not have any non-trivial automorphisms. Surprisingly, it turns out that any such tree can be explored by only 4 robots. This shows that the difficulty in tree exploration is mainly due to symmetries existing in the tree. (This is also somehow confirmed by the fact that the lower bound in Section 3 was shown for the complete binary trees, which have a lot of symmetries). We also show that the bound of 4 robots is tight for the class of rigid trees.

4.1. Algorithm Rigid-tree-exploration

We first define the following total order $\preceq$ on the set of all rooted rigid trees. For any such trees T and T', if the height of T is smaller than the height of T' then $T \preceq T'$. For trees of the same height h the order is defined by induction on h. There is only one rooted rigid tree of height 0 and of height 1. Suppose that all trees of height $h' < h$ are ordered, and consider trees T and T' of height h. Let $v_1, \ldots, v_k$ be children of the root of T, and $v'_1, \ldots, v'_m$ children of the root of T', such that subtrees $T_1, \ldots, T_k$ and $T'_1, \ldots, T'_m$ rooted at those nodes are in increasing order $\preceq$. Then $T \preceq T'$ if and only if the sequence $(T_1, \ldots, T_k)$ precedes $(T'_1, \ldots, T'_m)$ in the lexicographic order induced by $\preceq$.

Let T be any rigid tree. The tree T has either a centroid (node) v or a bicentroid (edge) e. In the first case root the tree at v and in the second case root the tree at the extremity of e whose corresponding rooted tree precedes the other in order $\preceq$. Arrange subtrees rooted at children of any internal node in increasing order $\preceq$. This induces a unique order of leaves of T. Call this order canonical, and let $1, \ldots, t$ be the enumeration of leaves of T in the canonical order. Hence each robot taking a snapshot of the tree at any time, perceives the same canonical order of leaves of the tree. Note that the distance between any two leaves of a rigid tree is at least 3.

The idea of the algorithm is the following. First the four robots occupy nodes forming a connected component containing leaf 1, then the robot from the neighbor of leaf 1 creates a tower in leaf 1. This tower is a signal to start exploration, which is done by visiting leaves in the canonical order by the two remaining robots. In the algorithm we use the notion of the robot closest to a given node. Ties are broken using order $\preceq$ as follows: of two robots situated at nodes v and w equidistant from u we choose the one that is in the earlier of the subtrees rooted at the children of the lowest common ancestor of v and w.

Algorithm Rigid-tree-exploration

Phase 1. There is no tower in the snapshot.
Let a_1 denote leaf 1 and a_2 its unique neighbor. If there is no robot in a_1 then the robot closest to a_1 goes toward a_1. If there is a robot in a_1 but no robot in a_2 then the robot closest to a_2 among robots outside a_1 goes toward a_2. If there are robots in a_1 and a_2 but no robot in other neighbors of a_2 then for the two remaining robots $\mathcal{R}_3$ and $\mathcal{R}_4$, let x_i denote the distance of robot $\mathcal{R}_i$ from the closest neighbor b_i of a_2 other than a_1. The robot $\mathcal{R}_i$ for which this distance is smaller goes toward b_i (ties are broken as above). If there are three robots in a_2, a_1 and in another neighbor of a_2 but the fourth robot is not a neighbor of any of them then this fourth robot goes toward the closest robot among the three. Finally, if all four robots form a connected component containing a_1 then the robot from a_2 goes to a_1 and creates a tower.

Phase 2. There is a tower in the snapshot.
If there is no free robot in a leaf, the robot closest to leaf 2 goes toward this leaf.

If there is exactly one free robot in leaf i then the other robot goes toward leaf $i+1$.
If there are free robots in leaves i and $i+1$, and $i+1$ is not the largest number of a leaf then the robot from i goes toward $i+2$.
If there are free robots in leaves i and $i+1$, and $i+1$ is the largest number of a leaf then all robots stop (explicit stopping condition).

Lemma 4. *Algorithm Rigid-tree-exploration is correct for any rigid tree with $n \geq 7$ nodes.*

PROOF. Consider the exploration of a rigid tree T. (Note that the smallest rigid tree with more than 4 nodes has in fact at least 7 nodes). Algorithm Rigid-tree-exploration has the property that at any time, at most one robot moves, for any execution of the algorithm. The fact that the algorithm stops at some time τ and that all leaves are explored at time τ follows from the formulation. It remains to show that all internal nodes are also explored at time τ. Notice that for any leaf i there is a (not necessarily simple) path P_i from leaf 1 to leaf i, all of whose nodes are explored at time τ. Consider any internal node v and let T_v be the subtree rooted at v. If leaf 1 is not in T_v then v must be on path P_i, for any leaf i in T_v. If leaf 1 is in T_v and v is not the root of T then there exists a leaf i outside T_v and v must be on path P_i, for any such leaf. Finally, if v is the root of T then consider the child w of v, such that leaf 1 is in the subtree rooted at w. Let i be a leaf outside this subtree. Then v must be on path P_i. It follows that v is explored at time τ, which concludes the proof. □

Theorem 4. *Any rigid tree with $n \geq 7$ nodes can be explored by 4 robots, starting from any initial configuration.*

4.2. Impossibility of exploration with 3 robots

We finally show that the size four of the team of robots used in Algorithm Rigid-tree-exploration cannot be decreased, except for very particular rigid trees.

Theorem 5. *Four robots are required to explore any rigid tree with at least four leaves.*

PROOF. Let T be a tree with at least four leaves. First assume that at least two nodes of T are of degree at least 3. Let v and v' be two nodes of degree at least three such that the nodes of the path from v to v' (if any) are of degree 2. Let T_1 and T_2 be two of the connected components of $T \setminus \{v\}$ not containing v'. Similarly, let T'_1 and T'_2 be two of the connected components of $T \setminus \{v'\}$ not containing v. If T has only one node u of degree at least 3, then this node has degree at least 4. We then set $v = v' = u$ and we define T_1, T_2, T'_1 and T'_2 as four different connected components of $T \setminus \{u\}$.

For the purpose of contradiction, let us consider a team of three robots able to explore (with stop) the tree T. Fix an initial configuration C_0. We consider an adversary that is sequential except for two situations. More precisely, each robot in turn completely executes its cycle Look-Compute-Move before the next robot proceeds with its own complete cycle. However, there are two exceptions. First, if a robot $\mathcal{R}_1$ is in a tower with exactly one other robot $\mathcal{R}_2$, executes the Compute operation based on its view of this configuration, and decides to move to a neighboring node occupied by the third robot $\mathcal{R}_3$, then the adversary makes the robot $\mathcal{R}_2$ act together with $\mathcal{R}_1$ so that both robots $\mathcal{R}_1$ and $\mathcal{R}_2$ move to the position of $\mathcal{R}_3$. Second, when all robots are at the same node, the adversary is synchronous, i.e., it never breaks the tower formed by the three robots. This adversary defines a sequence $S = (C_0, C_1, C_2, \cdots)$ of configurations starting from the initial configuration C_0.

Since the algorithm is supposed to be correct, the sequence S is finite: there exists an integer $f \geq 0$ such that all robots remain idle in the configuration C_f. This configuration contains a tower because otherwise the algorithm would fail to explore the tree starting from the legitimate initial configuration C_f. Since C_0 does not contain towers, we can define the unique index i such that C_i does not contain towers but all configurations C_j, for $i < j \leq f$, contain a tower. From now on, we consider C_i as the new initial configuration. This means that the tree must be explored completely during the sequence $S' = (C_i, C_{i+1}, \cdots, C_f)$. By the correctness of the algorithm we have $f > i+1$. It follows from the definition of S' and of the adversary that during the sequence S' a tower of two robots can move only to a node

occupied by a robot, hence forming a tower of three robots. Indeed, if a tower of two robots would move to an unoccupied node, then the adversary would move only one of the two robots, thus creating a configuration without towers and contradicting the definition of S'.

By definition of i, the configuration C_{i+1} contains a tower of two robots and the third robot $\mathcal{R}$ is located at some different node. At this point, at least two of the four subtrees under consideration are completely unexplored. Assume that $\mathcal{R}$ explores one of these unexplored subtrees, say T_1, before possibly joining (or being joined by) the two other robots, thus forming a tower of three robots. Robot $\mathcal{R}$ has to visit node v before entering T_1. Since the tower of two robots cannot move without the presence of $\mathcal{R}$ at its neighbor, this latter robot has to leave T_1, either to join the tower or to explore the remaining unexplored subtree by itself. To do that, it has to visit again node v. Since the configuration is exactly the same as the one just before entering T_1, robot $\mathcal{R}$ acts in the same way, i.e., reenters T_1. This means that the exploration cannot be finished.

Thus we can assume that $\mathcal{R}$ joins the two other robots before exploring any of the subtrees that are unexplored in configuration C_{i+1}. From now on, the three robots act simultaneously and the tower has still to explore at least two subtrees. Assume that it explores first T_1. Then the same argument applies: the tower visits node v before entering T_1, and then each time the tower tries to exit T_1, it visits v and thus reenters T_1. Hence, one subtree remains unexplored. This contradicts the assumption that there exists a successfull team of three robots, and concludes the proof of the theorem. □

Remark. The assumption in Theorem 5, concerning the number of leaves, cannot be removed. Indeed, it can be proved that any rigid tree with at most three leaves can be explored by three robots (but not by two robots).

5. Conclusion

In this paper we studied exploration of trees by a team of robots with contrasting strengths and weaknesses: unlimited visibility on one side and obliviousness and asynchrony on the other. The assumption we made of unlimited visibility has enabled us to focus on overcoming the other computational weaknesses of the robots in the design of our exploration algorithms, in particular the simultaneous presence of obliviousness and asynchrony. A natural next research step would be the investigation of the exploration problem when the visibility of the robots is limited, e.g., to the immediate neighborhood. Notice that, in this case, exploration is not generally possible. It is in fact easy to construct trees and place the robots in such a way that (1) initially they are all isolated (i.e., they have no other robots in their neighborhood), and (2) because of obliviousness, each robot will either never move or move forever back and forth across the same edge. Hence a limited visibility scenario could only work for some subset of initial configurations. Another line of research would be to equip robots with very small (e.g., constant) memory of past events and study how this additional power influences feasibility of exploration with limited or unlimited visibility. Finally, it would be interesting to extend our study to the case of arbitrary graphs, as well as to the stronger SSynch model.

References

[1] N. Agmon, D. Peleg, Fault-tolerant gathering algorithms for autonomous mobile robots, SIAM J. Comput. 36 (2006), 56-82.

[2] S. Albers and M. R. Henzinger, Exploring unknown environments, SIAM J. on Comput. 29 (2000), 1164-1188.

[3] H. Ando, Y. Oasa, I. Suzuki, M. Yamashita, Distributed memoryless point convergence algorithm for mobile robots with limited visibility, IEEE Trans. on Robotics and Automation 15 (1999), 818-828.

[4] I. Averbakh and O. Berman, A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree, Discr. Appl. Mathematics 68 (1996), 17-32.

[5] I. Averbakh and O. Berman, $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective, Discr. Appl. Mathematics 75 (1997), 201-216.

[6] B. Awerbuch, M. Betke, R. Rivest and M. Singh, Piecemeal graph learning by a mobile robot, Proc. 8th Conf. on Comput. Learning Theory (1995), 321-328.

[7] M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Proc. 30th Ann. Symp. on Theory of Computing (STOC 1998), 269-278.
[8] M.A. Bender and D. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, Proc. 35th Ann. Symp. on Foundations of Computer Science (FOCS 1994), 75-85.
[9] M. Betke, R. Rivest and M. Singh, Piecemeal learning of an unknown environment, Machine Learning 18 (1995), 231-254.
[10] M. Cieliebak, Gathering non-oblivious mobile robots, Proc. 6th Latin American Symposium on Theoretical Informatics (LATIN 2004), 577-588.
[11] M. Cieliebak, P. Flocchini, G. Prencipe and N. Santoro, Solving the robots gathering problem, Proc. 30th Int. Colloquium on Automata, Languages and Programming (ICALP 2003), LNCS 2719, 1181-1196.
[12] R. Cohen and D. Peleg, Local spreading algorithms for autonomous robot systems, Theoretical Computer Science 399(2008), 71-82.
[13] J. Czyzowicz, L. Gasieniec, and A. Pelc, Gathering few fat mobile robots in the plane, Proc. 10th International Conference on Principles of Distributed Systems (OPODIS 2006), LNCS 4288, 744-753.
[14] X. Deng and C. H. Papadimitriou, Exploring an unknown graph, Journal of Graph Theory 32 (1999), 265-297.
[15] A. Dessmark and A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (2004), 343-362.
[16] K. Diks, P. Fraigniaud, E. Kranakis and A. Pelc, Tree exploration with little memory, Proc. 13th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), 588-597.
[17] C.A. Duncan, S.G. Kobourov and V.S.A. Kumar, Optimal constrained graph exploration, Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2001), 807-814.
[18] G. N. Frederickson, M. S. Hecht and C. E. Kim, Approximation algorithms for some routing problems. SIAM J. on Computing 7 (1978), 178-193.
[19] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. Proc. 13th Annual European Symposium on Algorithms (ESA 2005), LNCS 3669, 11–22.
[20] P. Flocchini, D. Ilcinkas, A. Pelc and N. Santoro, Computing without communicating: Ring exploration by asynchronous oblivious robots, Proc. 11th International Conference on Principles of Distributed Systems (OPODIS 2007), 105–118.
[21] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, Arbitrary pattern formation by asynchronous, anonymous, oblivious robots, Theoretical Computer Science, 407(2008), 412-447.
[22] P. Flocchini, G. Prencipe, N. Santoro and P. Widmayer, Gathering of asynchronous robots with limited visibility, Theoretical Computer Science 337 (2005), 147-168.
[23] P. Fraigniaud, L. Gasieniec, D. Kowalski, A. Pelc, Collective tree exploration, Proc. Latin American Theoretical Informatics (LATIN 2004), LNCS 2976, 141-151.
[24] L. Gasieniec, A. Pelc, T. Radzik, X. Zhang, Tree exploration with logarithmic memory, Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), 585-594.
[25] R. Klasing, E. Markou and A. Pelc, Gathering asynchronous oblivious mobile robots in a ring, Theoretical Computer Science 390 (2008), 27-39.
[26] P. Panaite and A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (1999), 281-295.
[27] G. Prencipe, Impossibility of gathering by a set of autonomous mobile robots, Theoretical Computer Science 384 (2007), 222-231.
[28] G. Prencipe, N. Santoro, Distributed algorithms for mobile robots, Proc. 5th IFIP International Conference on Theoretical Computer Science (TCS 2006), 47-62.
[29] S. Souissi, X. Défago and M. Yamashita, Gathering asynchronous mobile robots with inaccurate compasses, In Proc. 10th Int. Conf. on Principles of Distributed Systems (OPODIS 2006), LNCS 4305, 333–349.
[30] K. Sugihara and I. Suzuki, Distributed algorithms for formation of geometric patterns with many mobile robots, Journal of Robotic Systems 13 (1996), 127-139.
[31] I. Suzuki and M. Yamashita, Distributed anonymous mobile robots: formation of geometric patterns, SIAM J. Comput. 28 (1999), 1347-1363.